10th Brazilian Congress on Computational Intelligence (CBIC'2011), November 8 to 11, 2011, Fortaleza, Ceará
Brazil

BASIC VNS WITH MEMORY APPLIED TO THE QAP

Abstract – This work deals with the use of memory within the basic structure of the VNS metaheuristic. The memory definitions are limited to the VNS neighborhood properties, avoiding the introduction of new parameters, thus preserving its simplicity. Some versions so built are tested with five instance classes of the Quadratic Assignment Problem (QAP). The resulting algorithms, with different forms of memory insertion, were compared among them and with the basic VNS with the aid of five differnt efficiency criteria.

Keywords – VNS, QAP, Metaheuristics, Computational Intelligence.

1 Introduction

The problems of discrete optimization are found in many applied situations and present generally a great number of alternative solutions. It is theoretically possible to enumerate every solution and evaluate it with respect to the desired objective, searching for the one(s) that give the best answer. This primary strategy is normally impossible to apply, because the number of solutions is frequently an exponential function of the problem size. A problem which typically presents this difficulty and which has been studied by several researchers is the Quadratic Assignment Problem (QAP), Koopmans e Beckmann [10]. We discuss its structure in Section 2.

Given the QAP's high degree of difficulty, there is a natural tendency favoring heuristics or metaheuristics intead of exact methods, which can be found in Loiola *et al* [11]. This happened mainly from the early 1990s on, resulting in the appearance of metaheuristics, which stimulated new research and resolution methods for combinatorial optimization problems.

The definition of general heuristic models, the metaheuristics, is a practice which has been adopted by a growing number of researchers, who use them as models to build specific heuristics for given combinatorial problems. This approach has the advantage of allowing the experience acquired with a problem to be considered when another problem is studied. This type of help proves to be more effective as more and more problems are submitted to a given metaheuristic and even to combined metaheuristics in the hybrid algorithms.

All metaheuristics contain escape techniques allowing the algorithm not to freeze in local optima. Since we can find significant differences between these local optima and the global one, the practical impact of these techniques has been very significative. In contrast with this practical success, the theoretical study of the metaheuristics has been difficult and the results emerge only gradually. Certainly, there are many good heuristics, built intelligently and that use several adjusting parameters, but the reasons for their success are not generally well known. The situation is even more complicated when hybrid algorithms are used. On the other hand, some metaheuristics are very simple and require few parameters, or none.

This work aims to present an improvement proposal for the VNS (Variable Neighborhood Search) metaheuristic (Hansen and Mladenovìc [7]). We study the effects of the use of memory structures within the basic VNS, keeping its simplicity and the unique VNS parameter. The algorithms thus designed are applied to a set of QAP instances and the results are compared with each other and with those of the basic VNS. With this in mind, Section 2 presents a formal QAP definition; Section 3 provides a succinct description of the VNS metaheuristic. Section 4 discusses the use of memory in VNS, which is the motivation for this work. Section 5 discusses the evaluation tool utilized, while presentation and discussion of the testing results are the object of Section 6. Our conclusions and suggestions for future work are found in Section 7.

2 The Quadratic Assignment Problem

Let us consider a situation concerned with activities and locations, where pairs of activities are allocated to pairs of locations, taking into account the costs associated with the distances and some flow of units between the activities. A number of practical situations can be modelled this way, from job shops to turbine paddles, and including hospital and ship layouts. The Quadratic Assignment Problem (QAP) consists of finding a minimum cost allocation of activities to locations, when the costs are given by the sum of distance-flow products.

Among the various available formulations of the QAP, we chose to use the permutational formulation proposed by Burkard and Stratman [2]. Let $X = \{1, 2, ..., n\}$, Π_n be the set of all X permutations, f_{ij} the flow between activities i and j, and $d_{\varphi(i)\varphi(j)}$, the distance between locations $\varphi(i)$ and $\varphi(j)$ given by the permutation $\varphi \in \Pi_n$. Each permutation φ thus corresponds to an allocation of exactly one activity to exactly one location and the objective function is then

$$z = \min_{\varphi \in S_n} \sum_{i<j}^{n} f_{ij} d_{\varphi(i)\varphi(j)} \qquad (1)$$

The data base is therefore a pair of matrices, that of distances $D = (d_{kl})$ e and that of flows, $F = (f_{ij})$, where the cost of allocating the facilities i, j simultaneously to the locations k, l is given by the product $f_{ij}d_{kl}$.

2.1 Characteristics of QAP instances

The QAP library, QAPLIB [13] contains about 140 instances that are frequently used by researchers to compare solution techniques. Their orders go from 12 to 256 vertices, but only four of them have an order greater than 100. Many of them are quite well resolved through heuristic techniques, and those based on metaheuristics are generally able to find solutions with deviating less than 1% from the best known results. The exact techniques have huge difficulties even to guarantee the optimality of the better known solutions, in some cases for instances of order less than 30. To date, no instance of order greater than 40 has been solved precisely.

Drezner *et al* [3] and Taillard [15] reported that some heuristic methods used with the QAP have shown that their efficiency depends strongly on the type of instance, whereby a method that is good for a given problem proved to be inefficient for another. This suggests that either the method needs to be adapted or a more suitable method must be found. They proposed some instances that are difficult to resolve with metaheuristics, chiefly using the most recent techniques built on pair exchange within the neighborhoods since they show local optima with values of over 4 times that of the global optimum, but that are easy to solve using exact methods. These instances are also applied here, increasing the problem bank to 168 instances.

QAP instances can be classified in five classes, as follows:

- Class 1 – Random instances with uniformly distributed distances and flows *(47 instances)*;
- Class 2 – Random flows in grids *(30 instances)*;
- Class 3 – Real-life problems *(38 instances)*;
- Class 4 – Random instances similar to real-life problems *(16 instances)*;
- Class 5 – Instances which are difficult to solve using metaheuristics *(37 instances).*

3 Variable Neighborhood Search

In 1997, Hansen and Mladenovìc [7] proposed a metaheuristic based on a systematic neighborhood associated with a random algorithm in order to find initial points for a local search, called Variable Neighborhood Search (VNS). In contrast with other methods based on local search methods, VNS does not follow a path, but incrementally explores neighborhoods more or less distant from the current solution. Some characteristics of the current solution can thus be used to build promising neighbor solutions. Using only one neighborhood may limit heuristic performance and, if the structure chosen is not suitable, the quality of the solutions may be impaired. The authors consider that the use of several neighborhoods provides very different local optima, which results in more opportunities for local searches.

Figure 1 shows the basic VNS algorithm, Hansen and Mladenovìc [8].

```
Initialization. Select the set of neighborhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{max}$, that will be used in
    the search; find an initial solution $x$; choose a stopping condition;

Repeat the following until the stopping condition is met:
    (1) Set $k \leftarrow 1$; (2) Until $k = k_{max}$, repeat the following steps:
    (a) Shaking. Generate a point $x'$ at random from the $k^{th}$ neighborhood of $x$ ($x' \in \mathcal{N}_k(x)$);
    (b) Local search. Apply some local search method with $x'$ as initial solution; denote with $x''$
    the so obtained local optimum;
    (c) Move or not. If this local optimum is better than the incumbent, move there ($x \leftarrow x''$),
    and continue the search with $\mathcal{N}_1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;
```

Figure 1 – Steps of basic VNS

4 Memory on basic VNS

The use of memory in metaheuristics incorporates part of the multi-memory model proposed by Atkinson and Shiffrin [1], based on the hypothesis for the way this process occurs with human beings. Our *short term memory* is used when we need to preserve information only for immediate use, as when we look a telephone number up in the phone book, make the call and then forget very quickly the number. But there is information that we store in our memory and can remember for decades. This kind of information has been associated with *long term memory*.

Melo *et al.* [12] considered two kinds of control by memory, both restricted to the present neighborhood and iteration.

- ***Control by vertex*** – if the vertex is used, its position cannot be changed during one or more exchanges;
- ***Control by movement*** – if an exchange is made, it cannot be reused during one or more exchanges.

These controls were inserted at two very well defined points in the basic VNS, as shown in Figure 1.

(a) Shaking memory in the present neighborhood

It enhances the future solution generation through the shaking ($x \rightarrow x'$) of the present neighborhood. It uses the same principle as the Tabu Search metaheuristic, Glover [5], [6], thus reducing the risk of cycling (return to a previous solution near the current one);

(b) Improvement memory at the present neighborhood

Its use concerns only the improvements obtained by the local search ($f(x'') < f(x)$), whereby the use of this information is preserved in generating the subsequent neighborhoods. This is equivalent to a crossover scheme from the Genetic Algorithm metaheuristic, Holland [9], where one tries to ensure that fathers' good traits are inherited by their sons.

These (binary) controls are applied until there is no longer random choices at the present neighborhood agitation, or until an improvement is obtained in that neighborhood, whichever occurs first. Melo *et al* [12] used these two mechanisms on single-control strategies applied to the basic VNS.

In the present work, we seek to evaluate the effect of double controls also based on those mechanisms. Following the strategies adopted in the reference, the present work also sought to obtain simplicity and code economy. For instance, when both controls were used the basic structure of control by movement (size $N = n(n - 1)/2$) can shelter that of control by vertex (size n). We then used a function to examine every movement associated with a given vertex, thus obtaining control by vertex.

By combining the two controls with the two possible control locations (agitation and present neighborhood improvement), we developed the following memory-based algorithms:

- Control by vertex, both in the agitation and in present neighborhood improvement – *DoubleVertVert*;

- Control by vertex in the agitation and by movement in the present neighborhood improvement – *DoubleVertMov*;
- Control by movement in the agitation and by vertex in the present neighborhood improvement – *DoubleMovVert*;
- Control by movement in the agitation and in the present neighborhood improvement – *DoubleMovMov*.

5 Methodology

We used here the same methodology as in Melo *et al*. [12], that is, the same neighborhood structures (defined below) and the same seeds for ten independent executions of each instance with each method, also working with QAPLIB, Drezner et al [3] and Taillard [15] instances. The implementations were also done in C language and all tests were conducted on a computer with an Intel Core 2 Quad 2.4GHz processor, 4Gb RAM memory, a Linux operating system and *openSUSE* distribution.

- ***First Neighborhood***
  This neighborhood uses the strategy suggested by Taillard[14]: It starts with the first vertex and performs a random exchange with one of the remaining *n - 1* vertices. The first vertex is fixed and the second vertex undergoes a random exchange with one of the remaining *n - 2* vertices. The second vertex is fixed , we do the same with the third vertex, and so on. This idea is quite straightforward and avoids cycling. If it finds an improvement, then the best solution is updated.
- ***Second Neighborhood***
  In this neighborhood, only the first element is chosen randomly. After this, we exchange its content with all other *n - 1* elements of the solution. If it finds an improvement, the best solution is updated.
- ***Third Neighborhood***

  Each vertex is exchanged with its successors in the vector ordering This structure is more computationally demanding than the others. Just as in the neighborhoods 1 and 2, if we find an improvement over the best solution, it is updated.

5.1 Performance criteria for the proposed versions

We used five comparison criteria among the proposed algorithms. They were designed to identify better results associated with lower criterion values.

a) ***Complement of the total number of optimal solutions obtained***

With this definition, lower values represent better results.

b) ***Average relative value distance***

This is the average of the values (obtained value - OVBKV) / OVBKV, for those tests not reaching the OVBKV, expressed in percentages. (OVBKV is the abbreviation for **O**ptimum **V**alue or **B**est **K**nown **V**alue).

c) ***Quality index***

This is a function used to express performance, whether the algorithm reaches the OVBKV or not. Our choice was

$$I(nsol,error) = nSolNotOtm * (AvgError\%) + nSolOtm^{-1}$$

where *nSolOtm* is the number of solutions with the OVBKV value, *nSolNotOtm* is the number of solutions with worse values and *AvgError* is the average error percentage of those instances where the OVBKV was not reached.

As an example, let us consider that in 10 executions of an instance, 8 produced the OVBKV and 2 presented an error of 1.3%. The index value is then I = (2 x 1.3) + 0.125 = 2.725. If the error of those two instances were 21.6%, we would have I = (2 *21,6) + 0,125 = 43,325. We can see that the index is sensitive to worse solutions and that its value decreases when the number of OVBKV solutions grows.

d) ***Average execution time***

This includes only the instances where OVBKV was obtained before the maximum execution time (600 seconds).

e) ***Average stagnation time complement***

This shows the average difference between the maximum execution time of 600 seconds and the time associated with the last improvement in the solution value before the algorithm stops by maximum time criterion. Whenever the algorithm gets the OVBKV, this value is null. This criterion could be associated with the quality of avoiding to stick at local optima.

5.2 Comparison method for the proposed versions

A global performance evaluation of the proposed algorithms, the Weight Ordering Method (WOM), was used. The WOM uses a matrix where each line corresponds to an algorithm and each column to a given order. Each entry (*i*,*j*) contains the number of times the algorithm *i* appeared in order *j*. The example presented in Table 1 (for Tai25a instance) shows that Algorithm 1 obtained one first position, three third positions and one fourth position.

Table 1 – WOM method – ordering matrix

Algorithm	Ordering Matrix				
	1o.	2o.	3o.	4o.	5o.
1 - BasicVns	1	0	3	1	0
2 - DoubleMovMov	0	2	0	1	2
3 -DoubleMovVert	3	1	1	0	0
4 - DoubleVertMov	1	1	1	1	1
5 - DoubleVertVert	2	1	2	0	0

We use this matrix to associate a weight function with the set of orders in order to quantify their importance for each algorithm. A first-rated algorithm has to be relatively more important than the algorithm rated in second place and so on. The weight function for a given instance class *i*considers the number *w* of algorithms, the order of the algorithm for a given instance, an exponent basis $k > 1$ and the matrix $O = [O_{ij}]$ of the instance.

This function is injective over the set of natural numbers, thus ensuring that each value set it receives produces a different function value. The *k* value, an integer, has to be chosen in such a way that the results can be easily evaluated, that is, the obtained weights do not have values in a narrow range. After some trials, we chose $k = 3$.

Taking as an example the Tai25a instance, Table 2 shows the results for the compared algorithms. The last column shows the corresponding function values. We can see that the best global performance was that of Algorithm 3 (279) and the worst, that of Algorithm 2 (59).

Table 2 – WOM method – final algorithm ordering

Algorithm	Ordering Matrix					WtFunc
	1o.	2o.	3o.	4o.	5o.	
1 - BasicVns	1	0	3	1	0	111
2 - DoubleMovMov	0	2	0	1	2	59
3 -DoubleMovVert	3	1	1	0	0	279
4 - DoubleVertMov	1	1	1	1	1	121
5 - DoubleVertVert	2	1	2	0	0	207

It is important to mention that these results are consistent only within a given situation since the orderings obtained in two different situations may not be consistent and it may not be possible to add up their respective ratings.

6 Computational Results

Starting with the algorithm number, Table 3 presents in column 2 (Qtt) the number of instances which obtained the OVBKV at least once. The second and third columns give the number of instances and the average error for the instances where the OVBKV was not attained but where the average error was less than 2%. The fourth and fifth columns give the same information for those cases with an average error greater than 2%. The following five columns show the results of the five comparison criteria already described in Section 4.1.

Table 3 – General performance data for the algorithms

	Obt	ErrorMed(<=2%)		ErrorMed(>2%)		10-OVBKV	(dist,%)	I(nsol,error)	(t,exec)	(t,stag)
Algorithm	Qtt	Qtt	Perc (%)	Qtt	Perc (%)	Total	Avg (%)	Avg (%)	Avg (%)	Avg (%)
1 - BasicVns	119	37	0,18	12	25,14	691	25,37	180,59	15,61	256,91
2 - DoubleMovMov	122	33	0,44	13	30,43	690	25,91	178,84	21,56	254,57
3 -DoubleMovVert	118	35	0,47	15	31,83	687	25,94	180,08	21,00	259,98
4 - DoubleVertMov	117	37	0,46	14	28,54	681	25,76	178,55	16,31	244,13
5 - DoubleVertVert	119	36	0,44	13	28,65	676	25,81	177,44	13,91	247,17

By looking at Table 3, we can discuss some points concerning the double-control memorized VNS.

The first control seems to be determinant for both the execution time and the stagnation time: when it is *Mov*, both times are greater than in those cases where it is *Vert*. Control by vertex seems to be more promising, on average, in terms of convergence. Also, the error percentages above 2% are greater with *Mov* than with *Vert*, since the second control has less influence.

The most robust version seems to be *DoubleVertVert*. Even if it did not improve the results from *BasicVNS* in terms of the OVBVK, it was better than the other versions for three out of the five criteria.

Table 4 shows the general ordering of algorithm evaluation for the five QAP instance classes defined in Section 2.1. The lines represent the instance classes. The position of each algorithm in the oredering for each instance class is indicated by the columns. The table entries show the algorithm number and its ranking as determined using the WOM method.

Table 4 – Algorithm ranking by instance class

	Algorithm ranking by instance (class)				
Classes	1o.	2o.	3o.	4o.	5o.
1	2 - 13869	5 - 13797	1 - 13181	4 - 12963	3 - 12727
2	1 - 9078	3 - 8644	2 - 8360	5 - 8128	4 - 7994
3	5 - 13274	4 - 13138	1 - 12880	2 - 12872	3 - 12574
4	4 - 3722	5 - 3696	1 - 3492	2 - 3294	3 - 3278
5	5 - 8871	3 - 8589	4 - 8251	2 - 8029	1 - 7975

Starting with the values from Table 4, we build Table 5, where the number of times each algorithm obtained a given ranking is registered in the corresponding column (for instance, Algorithm 2 – *DoubleMovMov*, ranked fourth three times). We conclude that Algorithm 5 – *DoubleVertVert* obtained the best results (two first and two second place rankings out of five). This confirms the results already visible in Table 4.

Table 5 – Algorithm ranking consolidation

	Ordering Matrix					
Algorithm	1o.	2o.	3o.	4o.	5o.	WtFunc
1 - BasicVns	1	0	3	0	1	327
2 - DoubleMovMov	1	0	1	3	0	297
3 - DoubleMovVert	0	2	0	0	3	171
4 - DoubleVertMov	1	1	1	1	1	363
5 - DoubleVertVert	2	2	0	1	0	**657**

As seen earlier, when the first control is *Mov*, the algorithms are less efficient and when it is *Vert*, they are more efficient. This seems, for good or bad results, to indicate robustness linked to the first control. The last column confirms this influence.

7 Conclusions

In this work, we have proposed using double memory controls in the basic VNS, in the interest of obtaining better algorithms. This proposal did not associate the memory control with any external parameter, with the present iteration or with the costs obtained for the succesive solutions thus linking it only to the basic VNS behavior. The new codes were tested with a collection of QAP instances.

Control by vertex proved to be more efficient. It had lower execution times and better quality indices. With the most difficult instances, those with error averages above 2%, it obtained better results than control by movement, the only advantage of which was that it accounted for a greater stagnation time. This characteristic points to a somewhat better capacity to depart from local optima.

Choice of the first control is important in terms of defining performance, and the *Mov* control is not as efficient as the *Vert* control.

If the time saved is what is important, the double control by vertex is certainly more productive. Vertex control imposes a strong restriction on random selection, but it can be beneficial to the local search, by directing the search to a more promising region.

8 References

[1] Atkinson, R. C., Shiffrin, R. M. (1968). *Human memory: A proposed system and its control processes*. In K.W. Spence and J.T. Spence (Eds.),The psychology of learning and motivation, vol. 8. London: Academic Press.

[2] Burkard, R. E., Stratman, R. H.(1978). *Numerical investigations on quadratic assignment problem*. Naval Research Logistics Quarterly 25, 129-140.

[3] Drezner, Z., Hahn. P.M e Taillard, E.D. (2005). *Recent Advances for the Quadratic Assignment Problem with Special Emphasis on Instances that are Difficult for Meta-Heuristic Methods*. Annals of Operations Research 139, 65-94,.

[4] Estany, C. P. (2010). *Table of prime numbers between 1 to 2,000,000*. Available from: http://pinux.info/primos/. Access in April 2010.

[5] Glover, F.(1989). *Tabu search-Part I* . ORSA Journal on Computing 1, 190-206.

[6] Glover, F.(1989). *Tabu search-PartII* . ORSA Journal on Computing 2, 4-32.

[7] Hansen, P., Mladenovic, N. (1997). *Variable neighborhood search*. Computers and Operations Research 24, 1097–1100.

[8] Hansen, P., Mladenovic, N. (2001). *Developments of Variable Neighborhood Search*. Les Cahiers du GERAD, G-2001-24.

[9] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[10] Koopmans, T. C., Beckmann, M. (1957). *Assignment problems and the locationsof economics activities*. Econometrica, vol. 25, pp 53-76.

[11] Loiola, E.M. (2000). *Um algoritmo com parâmetros estatísticos para o PQA*. Dissertação de M.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, Brasil.

[12] Melo, V. A. ; Leite, L. S. B. S. ; Boaventura Netto, P. O. (2009). . *O uso de memória no VNS Básico aplicado ao Problema Quadrático de Alocação*. Annals, XLI Simpósio Brasileiro de Pesquisa Operacional, 2026-2036, Porto Seguro, BA, Brazil.

[13] QAPLIB Home Page (2011). Disponível em: http://www.seas.upenn.edu/qaplib/, Acesso em abril de 2011.

[14] Taillard, E. (1991). *Robust taboo search for the quadratic assignment problem*. Parallel Computing 17, 443–455.

[15] Taillard, E. (1994), *Comparison of Iterative Searches for the Quadratic Assignment Problem.* Centre de Recherche sur les Tansports, Publication no 989.